DOCUMENTAȚIE

-Tema 2-

Detectarea și recunoașterea facială a personajelor din serialul de desene animate Familia Flinstone

Student: Rîncu Ștefania

January 21, 2024

Contents

1 Alegerea setului de date (exemple) pentru antrenare

Așa cum am observat și în cadrul laboratorului, primul pas în rezolvarea temei a constat în determinarea exemplelor pozitive și a celor negative pe care să le folosesc în etapa de antrenare a clasificatorului care să separe fețele de non-fețe.

Atât pentru selectarea exemplelor pozitive, cât și a celor negative, am folosit datele puse la dispoziție în directorul *antrenare*. Acesta conține pentru fiecare din cele patru personaje (Fred, Wilma, Barney, Betty) câte un director cu 1000 imagini de antrenare, iar fiecare imagine conține adnotate toate fețele personajelor care apar în ea.

1.1 Filtre de culoare

Pentru a optimiza procesul de detecție facială în cadrul primului task și pentru a extrage exemplele folosite în cadrul antrenării am adoptat o strategie bazată pe aplicarea unor **filtre de culoare**. Scopul acestei abordări este de a izola regiunile dintr-o imagine care conțin culoarea pielii sau nuanțe apropiate acesteia. Astfel, spațiul de posibilități care prezintă regiuni de interes (fețe) este semnificativ redus.

În cadrul algoritmului prezentat, am implementat două filtre de culoare, bazate pe paradigme diferite.

Primul filtru utilizat este un filtru de culoare aplicat în spațiul HSV (Hue Saturation Vaulue), delimitat de intervalul [0, 20] pentru nuanță, [20, 255] pentru saturație și [70, 255] pentru luminozitate.

```
skin_patch_hsv = cv.cvtColor(negative_example, cv.COLOR_BGR2HSV)
skin_patch = cv.inRange(skin_patch_hsv, (0, 20, 70), (20, 255, 255))
```

Figure 1: *Structura filtrului de culoare aplicat peste spațiul HSV folosit în detectarea nuanțelor pielii*

Astfel, sunt evidențiate regiunile din imaginea originală care prezintă nuanțe specifice pielii umane, cum ar fi tonurile de roșu, portocaliu și galben. De asemenea, prin selectarea pixelilor care au o saturație și o luminozitate moderată până la ridicată, sunt tratate diverse condiții de iluminare.

(a) (b)

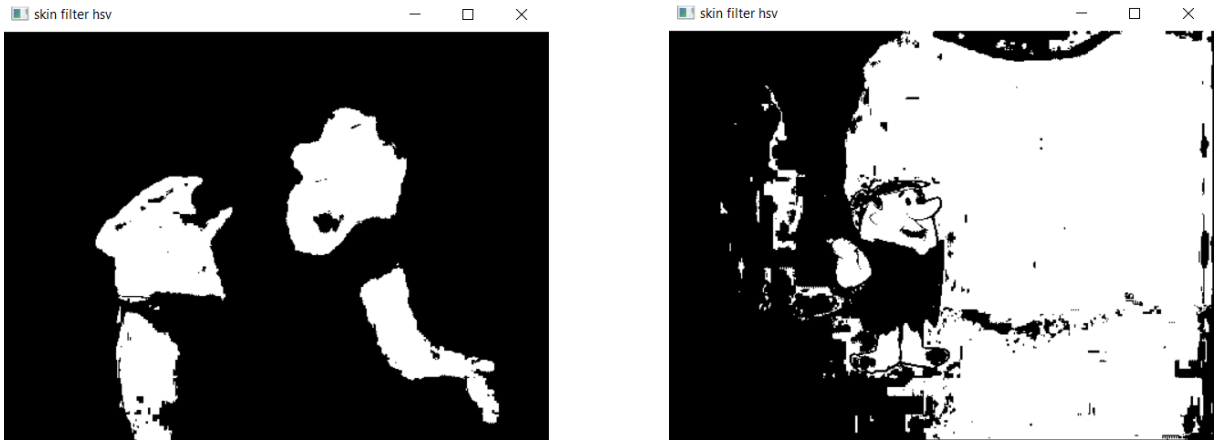


Figure 2: *Regiunile extrase prin aplicarea filtrului peste spațiul HSV descris mai sus*

După cum putem observa, filtrul de culoare prezentat mai sus este foarte eficient în detectarea

zonelor care conțin culoarea pielii în cadrul anumitor imagini (vezi: *Figura 2a*). Totuși, există situații în care fundalul prezintă nuanțe apropiate de cele ale pielii, ceea ce determină selectarea acestuia ca regiune de interes (vezi: *Figura 2b*).

Cel de al doilea filtru pe care l-am utilizat este bazat pe rezultatele găsite în mai multe articole care tratează problema detecției pielii în imagini. Acesta este un filtru mai complex, care folosește atât spațiul RGB, cât și spațiul HSV.

```
blue_frame, green_frame, red_frame = cv.split(resized_face.copy())

bgr_max = np.maximum.reduce([blue_frame, green_frame, red_frame])
bgr_min = np.minimum.reduce([blue_frame, green_frame, red_frame])

case_normal_light = np.logical_and.reduce([red_frame > 75, green_frame > 40, blue_frame > 20,
                                           bgr_max - bgr_min > 5, abs(red_frame - green_frame) > 5,
                                           red_frame > green_frame, red_frame > blue_frame])
case_artificial_light = np.logical_and.reduce([red_frame > 220, green_frame > 210, blue_frame > 170,
                                               abs(red_frame - green_frame) <= 15,
                                               red_frame > blue_frame,
                                               green_frame > blue_frame])

rgb_mask = np.logical_or(case_normal_light, case_artificial_light)

hsv_frame = cv.cvtColor(resized_face.copy(), cv.COLOR_BGR2HSV)
hsv_mask = np.logical_or(hsv_frame[..., 0] < 50, hsv_frame[..., 0] > 150)

final_skin_mask = np.logical_and.reduce([rgb_mask, hsv_mask]).astype(np.uint8) * 255
```

Figure 3: *Structura filtrului de culoare aplicat peste spațiile RGB și HSV*

În prima etapă se extrag din imagine canalele de culoare (albastru, verde, roșu), cu scopul de a oferi o perspectivă asupra distribuției acestora. Criteriile stabilite ulterior pentru delimitarea regiunilor care conțin piele în cazul diverselor condiții de iluminare sunt preluate din articolul [2] , și adaptate, prin micșorarea threshold-urilor astfel încât să corespundă cât mai bine imaginilor de antrenare și validare. În cea de-a doua etapă, am definit o mască în spațiul HSV, care păstrează pixelii cu valoarea nuanței sub 50 sau peste 150.

Figure 4: *Regiunile extrase prin aplicarea filtrului peste spațiile RGB și HSV descris mai sus*

Întrucât primul filtru modifica valorile intervalelor pentru luminozitate și saturație, acesta era

sensibil la variațiile de iluminare din poze. De aceea, multe fețe nu erau detectate în imagini mai întunecate.

După cum putem observa (vezi: *Figura 4*), cel de-al doilea filtru este mai permisiv față de cel definit anterior. De aceea, pe acesta îl folosesc în etapa de detecție.

1.2 Exemplele pozitive

Setul de exemple pozitive cuprinde **6920** de imagini de dimensiune **64x64** care conțin fețe. Procedeul de extragere al acestora a fost realizat cu ajutorul scriptului **„getPositiveExamples.py”**, în cadrul căruia, am folosit cele patru fișiere **.txt** de adnotații, pentru a prelua din fiecare imagine toate chenarele în care sunt încadrate fețe.

```
with open(f"{folder_path}{character}_annotations.txt", 'r') as file:
    for line in file:
        file_name, x_min, y_min, x_max, y_max, who = line.strip().split(" ")
        image_path = os.path.join(f"{folder_path}{character}", file_name)

        image = cv.imread(image_path)

        if image is not None:
            extracted_rectangle = image[int(y_min):int(y_max), int(x_min):int(x_max)]
            resized_face = cv.resize(extracted_rectangle, (64, 64))
```

Figure 5: *Secvența de cod care efectuează extragerea fețelor*

Pentru a **eticheta** fiecare personaj din exemplele pozitive și a simplifica ulterior task-ul de recunoaștere a acestora, am introdus o codificare a numelor princifre distincte. Astfel, **Barney - 0**, **Betty - 1**, **Fred - 2**, **Wilma - 3**, iar **Unknown - 4**.

Figure 6: *Exemplu de etichetare pentru exemple pozitive din clasele 2 și 3*

Înainte de a salva exemplele pozitive, am aplicat cel de-al doilea filtru de culoare menționat mai sus, astfel încât să păstrez un set de date care să respecte o anumită proprietate. Astfel, am eliminat fețe ale personajelor *unknown* (de exemplu, cele de culoare verde). De asemenea, am mai eliminat ulterior alte câteva imagini care erau mișcate pentru a reduce setul de date.

1.3 Exemplele negative

Setul de exemple negative cuprinde în varianta finală **207906** de imagini de dimensiune **64x64** care nu conțin fețe. Procedeul de extragere al acestora a fost realizat cu ajutorul scriptului **„getNegativeExamples.py”**, care are ca punct de pornire principiul prezentat în cadrul laboratorului, și anume, alegerea aleatoare a anumitor regiuni din imaginile de antrenare. În plus față de laborator, pentru rezolvarea temei a trebuit tratat cazul în care aceste regiuni intersectau fețele personajelor.

```
skin_patch_hsv = cv.cvtColor(negative_example.copy(), cv.COLOR_BGR2HSV)
skin_patch = cv.inRange(skin_patch_hsv, (0, 20, 70), (20, 255, 255))

if np.mean(skin_patch) >= 100:
    iou_with_faces = sum(intersection_over_union(current_bbox, np.array(bbox) * scale) for bbox in all_bboxes_file)

    if len(boxes_for_samples) > 0:
        iou_negative_samples = sum(intersection_over_union(current_bbox, np.array(bbox) * scale) for bbox in boxes_for_samples)
    else:
        iou_negative_samples = 0

    if iou_with_faces == 0 and iou_negative_samples < 0.339:
```

Figure 7: *Codul folosit pentru extragerea și filtrarea exemplelor negative*

Mai exact, preiau fiecare imagine din directorul *antrenare* și o scalez pentru a obține un set de date cât mai diversificat. Pentru **fiecare scală**, generez maxim **28** de exemple negative. De asemenea, pentru a lua în considerare doar zonele din imagine care conțin predominant culoarea pielii sau nuanțe apropiate acesteia, aplic **primul filtru de culoare** prezentat mai sus, care este mai restrictiv. Dacă media patch-ului extras după aplicarea acestui filtru depășește pragul impus, atunci verific dacă acesta **nu intersectează deloc fețele detectate** în acea imagine și dacă suma intersecțiilor cu exemplele negative deja extrase este suficient de mică. În caz afirmativ, salvez exemplul negativ.

2 Determinarea vectorilor de caracteristici pozitive

În cadrul metodei ***get_positive_descriptors***, prezentă atât în *FacialDetector.py*, cât și în *FacialRecognition.py*, am generat descriptorii pozitivi necesari pentru antrenarea clasificatorilor specializați în detecția și recunoașterea facială. Pentru a extinde setul de exemple pozitive, în plus față de **oglindirea** imaginilor, pe care am utilizat-o în cadrul laboratorului, am introdus două tehnici de augmentare suplimentare, respectiv **rotirea și translatarea** imaginilor.

```
for current_image in [img.copy(), img_flipped.copy()]:
    for angle in self.params.angles:
        rotation_matrix = cv.getRotationMatrix2D(center, angle, 1)
        rotated_image = cv.warpAffine(current_image.copy(), rotation_matrix, (width, height), borderMode=cv.BORDER_REPLICATE)

        for c in self.params.coord_recognition:
            translation_matrix = np.float32([[1, 0, c[0]], [0, 1, c[1]]])
            translated_image = cv.warpAffine(rotated_image, translation_matrix, (width, height),
                                             borderMode=cv.BORDER_REPLICATE)

            features = hog(translated_image, pixels_per_cell=(self.params.dim_hog_cell, self.params.dim_hog_cell),
                           cells_per_block=(2, 2), feature_vector=True)
            positive_descriptors.append(features)
            labels.append(int(files[i][-10]))
```

Figure 8: *Codul folosit pentru augmentarea si extragerea descriptorilor pozitivi*

În varianta finală, am păstrat doar **translațiile în sus, în jos, la stânga și la dreapta**, fără a mai include și mutările pe diagonală. Pentru fiecare imagine augmentată, am aplicat **algoritmul Histogram of Oriented Gradients (HOG)**, cu scopul de a extrage caracteristicile relevante.

Figure 9: *Vizualizarea descriptorilor HOG asociați anumitor imagini*

Funcția returnează descriptorii pozitivi și etichetele corespunzătoare, extrase din setul extins de exemple pozitive. În final, acesta conține un total de **207600** de elemente.

3 Determinarea vectorilor de caracteristici negative

În cadrul metodei ***get_negative_descriptors***, prezentă în *FacialDetector.py*, preiau exemplele negative din directorul specificat și memorez descriptorii HOG asociați, fără a mai augmenta setul de date.

4 Paradigma multi-scale

Pentru a aborda variabilitatea din distanța față de „cameră” (prim-plan) și a trata cazurile în care apar fețe mai apropiate sau mai depărtate și a adapta clasificatorul la aceste situații, am definit un vector care reține valori cuprinse între 1.5 și 0.4 (vezi: *Figura 10*), pe baza cărora redimensionez imaginea inițială.

```
scales = [1.5, 1.4, 1.3, 1.2, 1.1, 1.0, 0.9, 0.8, 0.7, 0.6, 0.5, 0.4]
```

Figure 10: *Vectorul de scale*

Folosind această paradigmă, am generat o piramidă de imagini redimensionate, precum cea ilustrată în Figura 11. Astfel, am îmbunătățit capacitatea clasificatorului de a detecta fețe în diverse condiții și distanțe.

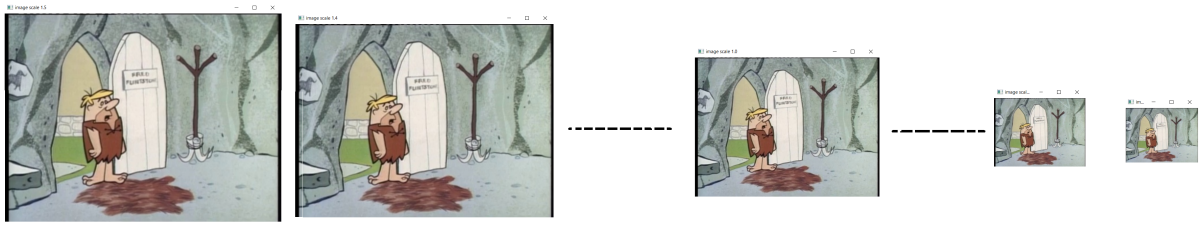


Figure 11: *Piramida imaginilor redimensionate cu o scală între 1.5 și 0.4*

5 Fereastra glisantă

Pentru a respecta cerința referitoare la implementarea unei soluții bazate pe paradigma **„sliding window”** pentru detectarea fețelor din cadrul acestei teme, am adoptat o abordare bazată pe explorarea secțiunilor succesive ale imaginii utilizând o fereastră de dimensiune **64x64**.

Inițial, păstrasem dimensiunile ferestrei din cadrul laboratorului, mai exact 36x36, dar am observat faptul că regiunile selectate sunt mult prea mici în raport cu dimensiunile chenarelor pe care le caut. În plus, folosind o fereastră de dimensiuni mai mari, am redus semnificativ timpul de testare.

Pentru a găsi un tipar în forma și dimensiunea bounding box-urilor pe care ar trebui să le detectez, am determinat o **medie a înălțimilor, lățimilor și al aspect ratio**-urilor pentru toate fețele preluate ca exemple pozitive.

```
barney
Mean height: 75.0, Mean Width: 92.0
Mean aspect ratio: 1.2266666666666666


betty
Mean height: 72.0, Mean Width: 75.0
Mean aspect ratio: 1.0416666666666667


fred
Mean height: 91.0, Mean Width: 84.5
Mean aspect ratio: 0.9285714285714286


wilma
Mean height: 91.0, Mean Width: 73.0
Mean aspect ratio: 0.8021978021978022
```

Figure 12: *Valorile medii obținute din analiza chenarelor exemplelor pozitive*

Pe baza acestor rezultate am inclus un vector (vezi Figura 13), cu ajutorul căruia să redimensionez imaginea originală pe orizontală sau verticală, astfel încât să obțin **existența unor bounding-box-uri dreptunghiulare**.

```
for xy_scales in [[0.81, 1], [0.96, 1], [1.08, 1], [1.25, 1], [1, 1.22], [1, 1.04], [1, 0.92], [1, 0.8]]:
    img_scaled = cv.resize(img.copy(), (0, 0), fx=scale*xy_scales[0], fy=scale*xy_scales[1])
    img_scaled_gray = cv.resize(img_gray.copy(), (0, 0), fx=scale*xy_scales[0], fy=scale*xy_scales[1])
```

Figure 13: *Valorile folosite pentru simularea dreptunghiurilor*

6 Procedeul folosit pentru detectarea fețelor

6.1 Etapa 1: Preluarea setului de date folosit în antrenarea modelului

Ca set de date pentru antrenarea clasificatorului de detecție am folosit vectorii ce conțin caracteristicile HOG extrase, conform detaliilor prezentate în secțiunile 2 și 3.

6.2 Etapa 2: Antrenarea clasificatorului

Pentru rezolvarea task-ului de detecție am ales să folosesc un model **MLPClassifier (Multi-Layer Perceptron Classifier)**, importat din biblioteca *sklearn.neural_network*

```
model = MLPClassifier(activation='relu', solver='adam', learning_rate='adaptive', max_iter=1500,
                      verbose=True, warm_start=True, n_iter_no_change=5)
```

Figure 14: *Clasificatorul folosit pentru detecție*

Pentru funcția de activare, am folosit **ReLU (Rectified Linear Unit)**, știind că aceasta elimină valorile negative, accelerând învățarea și reducând timpul de antrenare. Ca solver am ales **Adam**, cu scopul de a optimiza ponderile. Din proiectele trecute, știam că un learning rate constant nu favorizează învățarea modelului, de aceea l-am setat **adaptive**. Numărul maxim de epoci este egal cu **1500**, pentru a permite modelului să învețe de-a lungul unui număr suficient de iterații. De aseemenea, am configurat modelul astfel încât acesta să oprească antrenarea dacă nu observă o îmbunătățire într-un decurs de maxim **5** epoci consecutive. Cel mai bun model pentru detecție este salvat în fila **mlp_detection_207906_207600.npy**.

6.3 Etapa 3: Detectarea fețelor

În cadrul metodei *run_detection* implementată în script-ul *FacialDetector.py* procesez pe rând imaginile de testare. Procesul de detecție are la bază paradigma **multi-scale sliding window** menționată mai sus. Astfel, fiecare imagine este redimensionată cu aceeași valoare atât pe orizontală, cât și pe verticală, pentru a obține piramida de imagini. Totuși, cu scopul de a detecta și ferestre dreptunghiulare, aplic mici modificări și asupra lungimilor sau lățimilor separat (vezi secțiunile 4 și 5). Calculez descriptorii HOG pentru fiecare regiune din imagine și extrag. de asemenea, și patch-ul care îi corespunde în imaginea originală (color). Pentru a reduce numărul de exemple testate, și a maximiza șansele de a detecta doar regiuni din imagine care conțin piele, întâi aplic cel de al doilea filtru de culoare prezentat în secțiunea 1.1. Dacă acesta depășește un anumit prag al mediei de culoare, atunci trimit descriptorul HOG clasificatorului prezentat mai sus.

```
predictions = self.best_model_detection.predict_log_proba([descriptor])[0]
prediction = np.argmax(predictions)
score = np.max(predictions)

if prediction == 1 and score >= self.params.threshold:
```

Figure 15: *Interpretarea predicțiilor*

Ca **scor de încredere**, în cazul predicțiilor făcute de clasificatorul MLP, acesta întoarce logaritmii probabilităților prezise pentru fiecare clasă, adică valori cuprinse între 0 și 1. Aceste scoruri sunt returnate sub **formă negativă** fiind logaritmate. Pentru a determina dacă un patch este față sau non-față, folosesc acest scor de încredere, alegând clasa care are cel mai mare scor. De asemenea, pentru a elimina rezultatele fals pozitive, am introdus un threshold pentru scor.
În final, funcția returnează coordonatele detecțiilor (poziția bounding-box-urilor găsite pentru fețele din imagine), scorurile asociate acestora și denumirile filelor corespunzătoare imaginilor de test.

7 Procedeul folosit pentru recunoașterea fețelor

7.1 Etapa 1: Preluarea setului de date folosit în antrenarea modelelor

Primul pas este similar celui prezentat în secțiunea 6.1. În plus față de aceasta, am implementat și funcția *get_training_examples_each_character*, care se ocupă de preluarea seturilor de date necesare pentru antrenarea individuală a clasificatorilor destinați fiecărui personaj în parte (Barney, Betty, Fred, Wilma).

Pentru a determina cea mai bună **distribuție** a datelor de antrenare, am efectuat o analiză a situațiilor în care există cel mai des **confuzii** între personaje. De exemplu, am obervat faptul că Barney este adeseori încurcat cu Fred, iar Betty cu Wilma.

```
# betty
betty_faces_nr = len(dict_features_each_character[1])

random.shuffle(dict_features_each_character[3])
random.shuffle(dict_features_each_character[4])
random.shuffle(negative_features)

dict_train_each[1] = np.concatenate((np.squeeze(dict_features_each_character[1]),
                                     np.squeeze(dict_features_each_character[3][:int(betty_faces_nr * 0.4)]),
                                     np.squeeze(dict_features_each_character[4][:int(betty_faces_nr * 0.4)]),
                                     np.squeeze(negative_features[:int(betty_faces_nr * 0.2)])), axis=0)
dict_labels_each[1] = np.concatenate(
    (np.ones(betty_faces_nr), np.zeros(int(betty_faces_nr * 0.4) + int(betty_faces_nr * 0.4) + int(betty_faces_nr * 0.2))))
```

Figure 16: *Crearea setului de date pentru Betty*

Pentru a îmbunăți precizia modelelor, am **ajustat proporțiile seturilor de date** utilizate pentru antrenarea fiecărui clasificator. Astfel, am distribuit mai multe exemple pozitive și negative către personajele care au prezentat **confuzii semnificative**. În plus, amestec la fiecare pas vectorii de trăsătur cu scopul de a nu prelua mereu aceleași trăsături în aceeași ordine.

7.2 Etapa 2: Antrenarea clasificatoarelor

Pentru rezolvarea task-ului de recunoaștere am ales să folosesc câte un model pentru fiecare personaj în parte. În cadrul funcției *train_mlp_recognition_each* am definit un model **MLP (Multi-Layer Perceptron Classifier)** de bază, pe care îl antrenez utilizând seturile de date corespunzătoare fiecărui personaj.

```
model = MLPClassifier(activation='relu', solver='adam', learning_rate='adaptive', max_iter=1000,
                      verbose=True, warm_start=True, n_iter_no_change=5)

model.fit(train_examples, train_labels)

pickle.dump(model, open(recognition_nn_file, 'wb'))

print("Statictici pt modelul pt personajul -- " + str(character))
score = model.score(train_examples, train_labels)
print(score)

if character == 0:
    self.best_model_recognition_0 = model
elif character == 1:
    self.best_model_recognition_1 = model
elif character == 2:
    self.best_model_recognition_2 = model
else:
    self.best_model_recognition_3 = model
```

Figure 17: *Clasificatoarele folosite pentru recogniție*

Parametrii modelului, sunt aproape identici cu cei folosiți în cadrul task-ului de detecție, prezentat în secțiunea 6.2. După finalizarea antrenării, fiecare model este salvat, având în denumire codificarea personajului pentru care a fost antrenat.

7.3 Etapa 3: Recunoașterea fețelor

În cadrul funcției *run_recognition*, procesul de recunoaștere facială se desfășoară pe baza descriptorilor obținuți în etapa anterioară, cea de detectare a fețelor. Acești descriptori sunt transmiși fiecăruia dintre cei patru clasificatori specializați în identificarea personajelor principale: Barney, Betty, Fred și Wilma.

```
pred_class_0 = self.best_model_recognition_0.predict_log_proba([descriptors[i]])[0]
pred_class_1 = self.best_model_recognition_1.predict_log_proba([descriptors[i]])[0]
pred_class_2 = self.best_model_recognition_2.predict_log_proba([descriptors[i]])[0]
pred_class_3 = self.best_model_recognition_3.predict_log_proba([descriptors[i]])[0]

predictions = [np.argmax(pred_class_0) * np.exp(np.max(pred_class_0)), np.argmax(pred_class_1) * np.exp(np.max(pred_class_1)),
               np.argmax(pred_class_2) * np.exp(np.max(pred_class_2)), np.argmax(pred_class_3) * np.exp(np.max(pred_class_3))]

if sum(predictions) > 0:
    final_prediction_index = np.argmax(predictions)
else:
    final_prediction_index = 4
```

Figure 18: *Interpretarea predicțiilor și determinarea clasei finale*

Pe baza scorurilor (probabilităților) returnate de fiecare clasificator am implementat un sistem de vot ponderat. Astfel, determin ce personaj are cel mai mare scor de încredere, luând în considerare mai multe rezultate. În cazul în care toate cele patru modele au încadrat un descriptor în clasa 0, atunci ar trebui sa înseame că acesta este personaj necunoscut sau face parte din fundal. Rezultatele finale sunt organizate și stocate separat pentru fiecare personaj.

8 Rezultate obtinute pentru datele de validare

Alegerea celor mai bune modele a fost bazată pe rezultatele obținute pentru datele de validare puse la dispoziție.

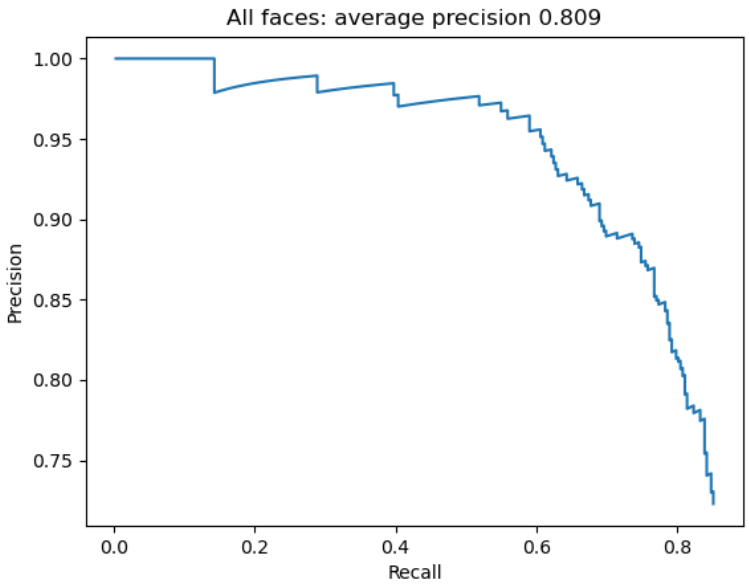
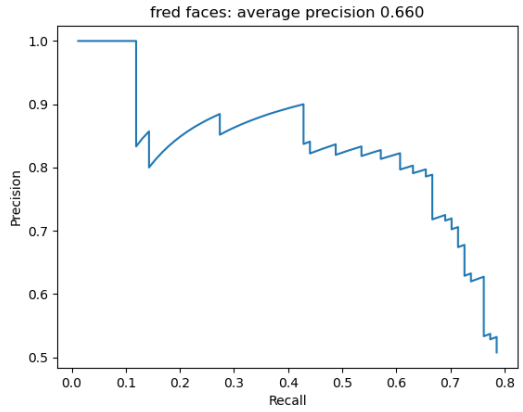
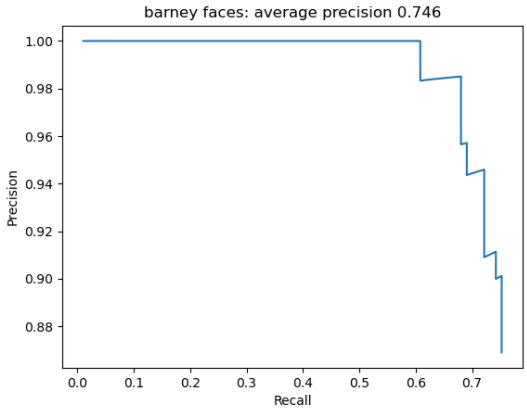


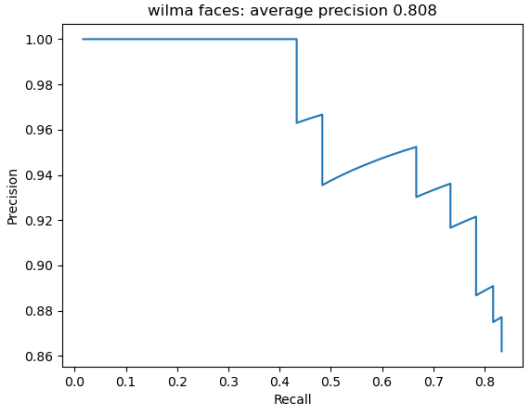
Figure 19: Precizia medie pentru task-ul de detecție



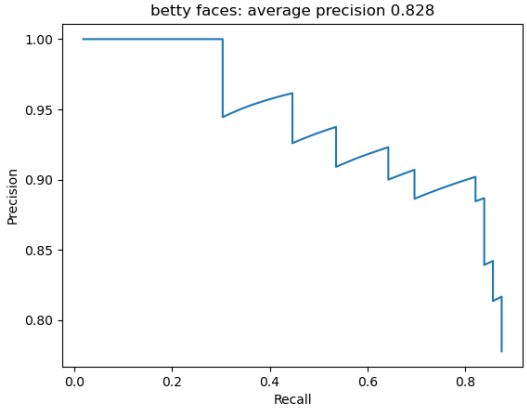
(a) Precizie medie Fred



(b) Precizie medie Barney



(c) Precizie medie Wilma



(d) Precizie medie Betty

Figure 20: Precizia medie pentru fiecare dintre cele 4 personaje (pe datele de validare)

9 Precizari

Algoritmul este destul de lent. Pentru a detecta și clasifica fețele din cele 200 de imagini puse la dispoziție în cadrul directorului *validare*, programul ruleaza între 4 și 5 ore.
În cadrul directorului *Cod_Sursa_Rincu_Stefania_332* am încărcat și scripturile folosite pentru generarea exemplelor pozitive și negative (*getPositiveExamples.py* și *getNegativeExamples.py*), chiar dacă acestea nu sunt folosite în cadrul programului. De asemenea, în fila *RunProject.py*, cu ajutorul căreia este rulat întreg proiectul, am păstrat codul folosit pentru antrenarea modelelor, cu toate că nici acesta nu este executat.

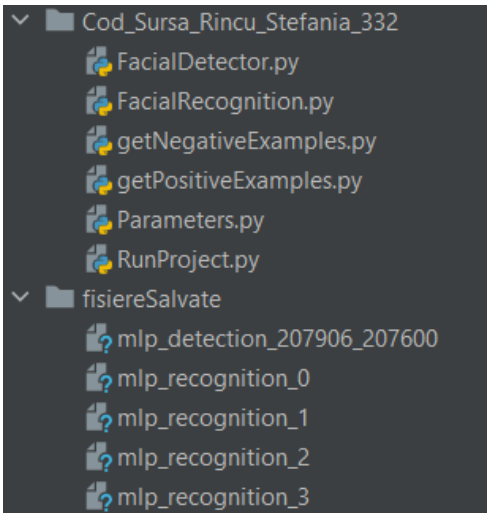


Figure 21: Structura folderelor încărcate

10 Referințe

[1] *Codul din cadrul laboratorului*

[2] *Human Skin Color Classification Using The Threshold Classifier: RGB, YCbCr, HSV(Python Code)*

[3] *Skin detection of animation characters*